

BASIC II

USER GUIDE



**ACORN
COMPUTER**

Acorn Computers Limited
Fulbourn Road, Cherry Hinton, Cambridge CB1 4JN.
Telephone 0223 245200

BASIC II

USER GUIDE

Part No. 980014
Issue 1
February 1984

Within this publication the term 'BBC' is used as an abbreviation for 'British Broadcasting Corporation'.

Copyright Acorn Computers Limited 1984

Neither the whole or any part of the information contained in, or the product described in, this manual may be adapted or reproduced in any material form except with the prior written approval of Acorn Computers Limited (Acorn Computers).

The product described in this manual and products for use with it, are subject to continuous development and improvement. All information of a technical nature and particulars of the product and its use (including the information and particulars in this manual) are given by Acorn Computers in good faith. However, it is acknowledged that there may be errors or omissions in this manual. A list of details of any amendments or revisions to this manual can be obtained upon request from Acorn Computers Technical Enquiries. Acorn Computers welcome comments and suggestions relating to the product and this manual.

All correspondence should be addressed to:-

Technical Enquiries
Acorn Computers Limited
Fulbourn Road
Cherry Hinton
Cambridge
CB1 4JN

All maintenance and service on the product must be carried out by Acorn Computers authorised dealers. Acorn Computers can accept no liability whatsoever for any loss or damage caused by service or maintenance by unauthorised personnel. This manual is intended only to assist the reader in the use of this product, and therefore Acorn Computers shall not be liable for any loss or damage whatsoever arising from the use of any information or particulars in, or any error or omission in, this manual, or any incorrect use of the product.

CONTENTS

Introduction	1
1. Alterations and extensions to BASIC keywords	2
ABS	2
COUNT	2
ELSE	2
EVAL	3
INPUT	3
INSTR	3
ON ERROR	4
OPENIN and OPENUP	4
2. Error handling	6
Fatal errors	6
Alterations to error messages	6
New error messages	7
3. Increases in precision and efficiency	8
LN and LOG	8
PRINT and STR\$	8
SIN and COS	8
String handling procedures	8
4. New commands	9
OPENUP	9
OSCLI	9
5. New features for assembler programmers	10
ASC	10
EQUB, EQUW, EQU D, EQU S	10
OPT	11
Appendix	13
Fitting the BASIC II ROM	13
Sideways ROMs - operating priorities	13
Removing the BASIC ROM	14
Inserting the BASIC II ROM	14
Illustration - Inserting the BASIC II ROM	15

1. Introduction

In many respects BASIC II is similar to BASIC; however there are several differences between the two. Some of the BASIC keywords have been extended so they are more powerful and additional ones have been incorporated. Furthermore several of the arithmetic routines have been recoded so that they give greater precision.

For the assembler programmer four new useful operations have been added and the 'OPT' instruction has been extended to enable programs to be relocated easily.

BASIC II will run on all BBC Microcomputers which have the operating system OS 1.0 or greater. To determine which operating system you have in your machine type

*HELP

The way to quickly distinguish BASIC II from BASIC is to type

REPORT

immediately after turning the machine on. BASIC II will give the message

(C)1982 Acorn

whereas BASIC will print

(C)1981 Acorn

BASIC II is now supplied automatically in BBC machines currently being produced. This pack is for people who have the ordinary BASIC and wish to take advantage of the extra facilities which BASIC II offers. These extra facilities are described in the pages that follow.

2. Alterations and extensions to BASIC keywords

ABS

The unary minus operator may be used in BASIC II, eg

```
PRINT -ABS(1)
```

This will now give the value '-1'

In BASIC this gave a 'Type mismatch' error.

COUNT

This counts the number of characters printed using 'PRINT', since the last new line. This has now been altered so that 'COUNT' is reset to zero after a change of mode, eg

```
10 PRINT "Hello";  
20 MODE 3  
30 PRINT "Goodbye";  
40 PRINT COUNT
```

In BASIC this would leave the screen showing:

```
Goodbye      12
```

With BASIC II the following is obtained:

```
Goodbye      7
```

ELSE

In BASIC 'ON ... GOTO / GOSUB ... ELSE' could not be used inside procedures or functions. Only the 'ON ... GOTO / GOSUB' part was available. Attempting to use an 'ELSE' part caused an error message to be printed. In BASIC II the 'ON ... GOTO / GOSUB' has been extended so that it can be used with an 'ELSE' part anywhere in a program.

Example

```
10 PRINT "If you want to know what all the  
little piggies were doing this afternoon, give  
the number of the little piggy you are  
interested in when asked."  
20 INPUT "Which little piggy",A  
30 PROCPIG(A)  
40 GOTO 20  
50 DEFPROCPIG(X)  
60 ON X GOTO 70,80,90,100,110 ELSE GOTO 120  
70 PRINT "The first little piggy went to
```



```

market.": ENDPROC
80 PRINT "The second little piggy stayed at
home.": ENDPROC
90 PRINT "The third little piggy had roast
beef.": ENDPROC
100 PRINT "The fourth little piggy had none.":
ENDPROC
110 PRINT "The fifth little piggy ran all the
way home." : ENDPROC
120 PRINT "There were only five little
piggies.": ENDPROC

```

EVAL

This function has been extended. In BASIC, 'EVAL A\$' could be used to evaluate strings such as:

```
A$ = "SIN(X/120) + COS (X/30)"
```

```
A$ ="PI"
```

In addition, in BASIC II the 'EVAL' function can be used to evaluate pseudo-variables such as 'TIME', 'PAGE', 'HIMEM' and 'LOMEM', eg

```
A$ = "TIME"
PRINT EVAL (A$)
```

will print out the current value of the pseudo variable 'TIME'.

INPUT

If more than one string or value is to be input at a time then the variable identifiers have to be separated from each other. In BASIC this was done using commas, eg

```
INPUT NAME$, AGE, HEIGHT
```

In BASIC II either commas or semicolons may be used, eg

```
INPUT NAME$, AGE; HEIGHT
```

When entering values, however, these should be separated by commas as before, not semicolons, eg

```
MICHAEL,21,170
```

INSTR

In BASIC, 'INSTR' could be used to find the occurrence of one string inside another, eg

```
PRINT INSTR("Hello","l")
```

would print '3' since the first 'l' is in the third character position. However, the second entry had to be shorter than the first for the function to operate correctly, eg

```
INSTR("l","Hello")
```

would not work correctly inside a function or procedure.

In BASIC II, 'INSTR' has been extended so that if a longer string is searched for inside a shorter one, as in the example, then this will result in 'INSTR' returning the value '0'.

This means that it is now possible to use, for example

```
100 INPUT "What is the longer string ",A$
110 INPUT "What string do you wish to search
    for",B$
120 X = INSTR(A$,B$)
```

inside a procedure or function without having to check that A\$ is longer than B\$.

ON ERROR

In BASIC it was possible to jump to most line numbers using the command 'ON ERROR GOTO ...', but not all lines could be jumped to in this way, for example 'ON ERROR GOTO 9999' could not be used. This command has now been altered to accommodate all line numbers.

OPENIN and OPENUP

In both BASIC and BASIC II an existing file can be opened to allow data to be read or altered, or to allow more data to be added to the end. In BASIC, this function was performed by the instruction 'OPENIN'; in BASIC II it is done by 'OPENUP'. Since these keywords give exactly the same result, the token for them both is &AD. Hence, if a program containing the instruction 'OPENUP' is written on a BBC Microcomputer containing BASIC II then the instruction will become tokenised to &AD. If this program is then saved and loaded onto a machine containing BASIC, the program will work in exactly the same way, but when listed, it will display the instruction as 'OPENIN'. This will apply the other way round as well so existing programs do not need to be altered to run in BASIC II.

The keyword 'OPENIN' does exist in BASIC II but it has

a different meaning. BASIC II uses the keyword 'OPENIN' to open a file for read-only operations; this was not possible in BASIC. Since this is a new facility it has a new token, &8E. Note that programs written in BASIC II which contain the instruction 'OPENIN' will not run in BASIC. This applies to all programs containing any of the new instructions.

Examples

```
10 DIM A$(20)
20 channel = OPENIN("name")
30 FOR N = 1 TO 20
40 INPUT#channel,A$(N)
50 NEXT
```

```
10 channel = OPENUP("name")
20 FOR N = 1 TO 20
30 P = PTR#channel
40 INPUT#channel,A$
50 IF A$ = "Jim" THEN PTR#channel = P :
PRINT#channel,"Joe"
60 NEXT
```

2. Error handling

The standard error handling procedures in BASIC II do not use any space on the software stack. This means that when a program runs out of all free space the error message printed out is no longer followed by a 'No room' message.

Fatal errors

'STOP' and 'No room' are now classed as errors, they have an error number of 0. They are, however, 'fatal errors' in that when they are processed they have an 'ON ERROR OFF' effect, eg

```
10 ON ERROR GOTO 30
20 STOP
30 PRINT "HELLO"
```

This will result in 'STOP at line 20' being printed since the error is not trapped.

Any error with error number 0 is classed as a fatal error in BASIC II. Hence fatal errors can now be generated by the user in assembler, eg

```
10 ON ERROR PRINT "An error has occurred"
.....
100 [
110 .stophere BRK
120          EQUB 0
130          EQU$ "Stop here"
140          EQUB 0
150 ]
.....
200 CALLstophere
```

See page 10 for an explanation of EQUB and EQU\$.

The call of 'stophere' will result in 'Stop here' being printed. However, if line 120 is replaced by

```
120          EQUB 20
```

then the call of 'stophere' will result in 'An error has occurred' being printed. This is because originally the error is given the number 0 and so is treated as a fatal error and is not trapped by the 'ON ERROR' command. In the second case, however, the error is given the number 20 and is treated like a normal error.

Alterations to error messages

The message printed out by the command 'STOP' has also been changed to 'STOP'. In BASIC it printed 'STOP at line 0'.

An illegal DIM statement, for example DIM P% -2, will now give the error message 'Bad DIM'.

New error messages

A new error message has been introduced, error 45 "Missing #".

This is printed if any of the following are used without a '#' sign:

PTR, EOF, BGET, BPUT, EXT.

3. Increases in precision and efficiency

LN and LOG

These functions have been rewritten to make them more accurate. This has a greater effect at the limits of the number range allowed.

PRINT and STR\$

With these functions in BASIC II, ten figures of precision on printing are allowed instead of nine. This allows the maximum positive integer 2147483647 to be printed out, eg

```
X = 2147483647
@% = &00000A0A
PRINT X
```

In BASIC the result is 2.14748365E9

In BASIC II the result is 2147483647

To retain compatability with 'PRINT', 'STR\$' has been given the default value of ten figures. This will result in 'STR\$' giving different values from before, eg 7.7 which is a recurring binary fraction will be converted to 7.699999999

SIN and COS

These have been recoded in order to increase their accuracy.

String handling procedures

The allocation of space for strings is more efficient in BASIC II, eg

```
REPEAT A$ = A$ + "*" : UNTIL LEN A$ = 255
```

In BASIC this would have allocated a total of 3882 bytes, whereas in BASIC II only 263 bytes are used. However, the optimisation is only implemented when just one string is being used at a time, eg

```
REPEAT A$ = A$ + "*" :B$ = B$ + "!" :
UNTIL LEN A$ = 255
```

In both BASIC II and BASIC this uses 7764 bytes.

4. New commands

OPENUP

See 'OPENIN and OPENUP' in chapter 1.

OSCLI

'OSCLI' takes a string expression and gives it to the operating system command line interpreter. Hence it acts in a similar way to the operating system commands, eg

```
OSCLI"CAT"
```

acts like

```
*CAT
```

However OSCLI is more powerful than this, eg

```
A$ = "CAT"  
OSCLI A$
```

is also equivalent to '*CAT' whereas

```
A$ = "CAT"  
*A$
```

cannot be used.

'OSCLI' is particularly useful for loading files, eg

```
OSCLI "LOAD " + FILE$ + " " + STR$(TOP-2)
```

will load a BASIC program onto the end of another BASIC program.

The 'STR\$' in the above example is a function which takes the value of TOP-2 as a number and returns the equivalent string. The '~' sign means that the hexadecimal representation of the number is taken rather than the decimal representation.

```
*LOAD FILE$ TOP-2
```

cannot be used instead of the OSCLI command since the '*LOAD' needs the address to be a hexadecimal value and will not accept a variable such as 'TOP-2'.

The shortest abbreviation for 'OSCLI' is 'OS.', and its token value is &FF. It has no unique errors of its own, just the normal 'Type mismatch' error.

5. New features for assembler programmers

ASC

In BASIC II 'ASC ":" may be used in the assembler. In the original BASIC this led to confusion.

EQUB, EQUW, EQUQ, EQU\$

These four operations are available in the assembler. They all take a single argument and put its value into the assembly code. The last letter in the name determines whether it is a byte, word, double word or string that it enters. 'EQU\$' puts all the characters of a string into the code without a carriage return; if one is required this should be added to the string itself. An example of the use of 'EQU\$' is shown below:

```
10      DIM CODE 200
20      oswrch = &FFEE
30      nop   = &EA
40      addr  = &70
50      FOR pass = 0 TO 3 STEP 3
60      P% = CODE
70      [
80      OPT pass
90.go    JSR vstring
100     EQU$ "hello"
110     NOP
120     RTS
```

.....

```
200.vstring    PLA : STA addr
210            PLA : STA addr + 1
220            LDY #0
230.vloop      INC addr
240            BNE nocarry
250            INC addr + 1
260.nocarry     LDA (addr),Y
270            CMP # nop
280            BEQ out
290            JSR oswrch
300            JMP vloop
310.out        JMP (addr)
320            ]
330            NEXT pass
340            CALL go
```

In the above program 'EQU\$' inserts all the characters from the string "hello" into the machine code. 'JSR vstring' then prints out all the characters until it reaches a 'NOP'.

'EQUUS' can also be used within assembler to generate macros, eg

```
10 FOR pass = 4 TO 7 STEP 3
20 [
30 OPT pass
.....
100 EQUUS FNADD(X,Y)
.....
200 ]
210 DEF FNADD(X,Y)
220 [
.....
300 ]
310 = ""
```

This would place the code generated by the function 'FNADD' in the main body of the code where the function was called. The '=' is there to signal the end of the macro since it ends the function call by returning a null string.

The four operations listed above have no errors of their own apart from the 'Type mismatch' error.

Note that 'EQUB', 'EQUW', 'EQUd' and 'EQUUS' have to be in upper case to be recognised.

OPT

In BASIC the lowest bit of the 'OPT' statement's operand determines whether the assembled code is listed or not and the second bit is used to suppress or report errors. In BASIC II the third bit is used as well to determine whether the code is put at 0% (the code origin) or P% (the program counter). If the bit is set then the code will be put at 0% and both 0% and P% will be incremented. If it is unset only P% will be used. For example 'OPT 4' (100 in binary) will suppress assembler errors, give no listing and will put the code at 0% but with all references referring to the locations they would be pointing to if the code were located at P%. This means that it would then be straightforward to relocate the code to P% and run it.

In BASIC only the first and second bits were relevant so setting the third bit had no effect, therefore the code was always put at P%.

Example

```
10 REM This compiles a program to produce code
to run at &400 which is in BASIC workspace so
direct assembly is not possible.
```



```

20 FOR pass = 4 TO 7 STEP 3
30 O% = &2000
40 P% = &400
50 [
60 .score      EQU 0
70             OPT pass
80 .scoreadd    TXA          \The Lo byte of
90             CLC           \the score to be
100            ADCscore      \added is stored
110            STAscore      \on X
120            TYA
130            ADCscore + 1   \The Hi byte of
140            STAscore + 1   \the score is
150            BCCnocarry     \stored in Y
160            INCscore + 2
170 .nocarry    RTS
180 ]
190 NEXT pass

```

The above assembly code is a routine to increment the score in a game and is intended to be part of a larger program which initialises the score and then jumps to the subroutine '.scoreadd' whenever it is required. When run, the program will compile the machine code generated to &2000 but it has to be run at &400 since the address references are fixed to point into this area.

To save the program you need to know the length of it. To find this, type

```
PRINT ~O%-&2000
```

This will print out the number of bytes of object code produced by the program. If this was &28A for example you could then type

```
*SAVE Score 2000 228A 400 400
```

where &228A = &2000 + &28A

Alternatively the 'OSCLI' command described earlier could be used:

```
OSCLI "SAVE Score 2000 " + STR$(O%) + " 400 400"
```

In either case to run the program type

```
*RUN Score
```

Fitting the BASIC II ROM

To use BASIC II you should remove the BASIC ROM from your machine and insert the BASIC II ROM into one of the free sockets. The ROM sockets are located on the front right-hand side of the circuit board inside the BBC Microcomputer casing.

1. To get to the board, undo the four screws which hold the casing together - on some computers these will be marked 'FIX'. Two of these screws are underneath the computer, and the other two can be found on the back.
2. Once the top is removed, release the bolts holding down the keyboard assembly. These are located on either side of the keyboard. Some machines have two bolts, others may have three.
3. There is no need to disconnect the keyboard completely, so the multi-wire connector to the main board can be left in place. Carefully displace the keyboard, rotating it clockwise through about 20 degrees so that the front right-hand side is accessible.
4. Locate the row of five large sockets. The one on the left contains the operating system. The BBC BASIC ROM should be in one of the other four. It can be identified by looking at the second line of writing which is a series of letters and numbers ending in the sequence 'B01'.

Sideways ROMs - operating priorities

The four sideways ROM sockets have an operating priority, decreasing from right to left; on a hard reset, or when the computer is switched on, the language chip in the rightmost ROM socket takes priority over the others. So the position of the BASIC II ROM in relation to any other languages present will determine whether your machine starts up in BASIC II or in another language.

If you want to start up in BASIC II then you must insert your BASIC II chip to the right of all other chips.

Removing the BASIC ROM

To avoid bending any pins the ROM must be removed very carefully. Take a screwdriver or similar tool and gently prize up each end, a bit at a time.

Inserting the BASIC II ROM

1. Before taking the ROM out of its protective packaging, identify Pin 1 on the ROM. It is either marked with a dot on the top, in the corner of Pin 1, or the half-moon notch at one end of the ROM identifies the end of the ROM nearest Pin 1. Pin 1 should be on the left if the notch is held up.

2. Hold the ends of the ROM between finger and thumb, and line up all the pins over the destination socket (see diagram). Pin 1 and the half-moon notch should point towards the back of the computer casing.

3. Now apply firm pressure to the ROM, but try not to force it! When the ROM is in, it appears to be slightly raised. Check that all the pins do enter the socket and that none are bent out or underneath.

Inserting the BASIC II ROM

This diagram shows a plan view of the BBC Microcomputer. The top of the computer casing has been removed to reveal the four sideways ROM sockets. The BASIC II ROM can be inserted in any one of these sockets.



